CACHING MECHANISM TO OPTIMIZE A BIDDING PROCESS USED TO SELECT RESOURCES AND SERVICES

FIELD OF THE INVENTION

The present invention relates in general to multi-agent resource allocation bidding systems and more particularly to a caching mechanism for storing most recent bids by bidders for a given bidding context in order to optimize the bidding process.

BACKGROUND OF THE INVENTION

Telecommunication systems have recently been designed for providing a variety of real-time services and features in an open distributed environment through the collaboration of a set of software components called *agents*. Such multi-agent systems are designed in such a way that they may adapt and evolve in the face of changing environments. One such multi-agent system is known as MANA (Multi-Agent Architecture for Networking Applications) developed by Mitel Corporation. Through the use of a distributed agent architecture, the system meets high reliability levels and adapts to accommodate technological or service evolution. To achieve these goals, intelligence or learning mechanisms are provided to update service information derived from the operation of the agents. This information is used to redefine the agents and to reallocate resources for correcting failures and to meet the requirements of a defined service more precisely.

An application or service in a multi-agent system is mapped as a series of calls amongst agents to perform the service. Each agent specifies its type, quantity and quality of service (QoS) in order to provide for an overall application. Since multi-agent systems are implemented in an open environment, no agent has prior knowledge of any other agent. The only knowledge that an agent possesses is its requirements and capabilities to provide a specific type of service. Thus, an agent may be required to find other agents to fulfill certain of its service requirements. A calling agent (referred to herein as a Bid Manager) sends out a bid for services to a plurality of called agents (referred to herein as Bidders), each of whom may be capable of providing the necessary resources for the Bid Manager to complete its task. The Bid Manager receives and evaluates the bids from the various Bidders and selects the

agent which has the best chance of success in performing the requested service. This is referred to herein as selecting the "lowest bid".

One example of a classic bidding mechanism of the foregoing type is disclosed in commonly owned U.S. Patent 5,675,636 entitled Adaptive Method of Allocating Calls. According to this system, a Router Agent sends a call for bids to a plurality of Carrier Agents in order to determine the cost to complete a call by each of the Carrier Agents. The system then selects the cheapest bid. In the embodiment disclosed in U.S. Patent 5,675,636 all of the Bidders are running in the same PBX as the BidManager. However, in the case of a distributed telecommunications system, some of the bidding Carrier Agents may be running on one or more remote PBX's that are linked with the local PBX through a leased line. In such a system it is important to optimize the bidding process to avoid having to request bids from each Bidder each time a service is requested. Requesting bids can be time-costly and therefore inapplicable in real-time applications such as Advanced Automatic Route Selection (AARS) as set forth in U.S. Patent 5,675,636, especially if the system is a distributed one.

SUMMARY OF THE INVENTION

According to the present invention, a caching mechanism is provided for storing the latest bids by one or more Bidders for a given bidding context. The cache is updated using a notification mechanism. The cache can also be consulted by the Bid Manager for some Bidders past the bidding deadline so that the Bid Manager does not lose a potentially good bid.

BRIEF DESCRIPTION OF THE DRAWINGS

An embodiment of the invention is described herein below with reference to the accompanying drawings, in which:

Figure 1 is a use case diagram showing use of the system by various external actors;

Figure 2 is a class diagram showing the internal representation of an agent in terms of a Feature which uses a ResourceManager and a ResourceAdapter selected by the ResourceManager;

Figure 2A is a class diagram with design patterns wherein a BidderAgentAdapter plays the role of adapter in the Adapter Pattern;

Figure 2B is a class diagram with design patterns wherein the BidderAgentAdapter invokes one of either a CachingAdapter or a NoCachingAdapter according to the Strategy Pattern;

Figure 3 is a block diagram showing selection between the CachingAdapter and the NoCaching Adapter;

Figure 4 shows pseudo-code for implementing a getBidValue() method of the CachingAdapter and the NoCachingAdapter;

Figure 5 shows pseudo-code for implementing selectAdapter() and sortAdapters() methods of the ResourceManager;

Figure 6 is an interaction diagram showing how Bidders decide on which kind of ResourceAdapter the BidManager is to use for the bidding protocol;

Figure 7 is an interaction diagram showing the case where the Bid Manager does not have any previous bid from a certain Bidder;

Figure 8 is an interaction diagram showing the Bid Manager caching a Bid value for the certain Bidder for a given context;

Figure 9 is an interaction diagram showing the Bid Manager sending a call for bid to a Bidder for whom there is no cached Bid value;

Figure 10 is an interaction diagram showing notification of a change in bidding values of a Bidder to the Bid Manager;

Figure 11 is a flowchart showing the steps of a method for implementing the caching system of the present invention; and

Figure 12 is a schematic illustration of an example of the caching system of the present invention applied to distributed advanced automatic route selection.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The Figures and description herein of a preferred embodiment of the invention follow the ODP (Open Distributed Processing) principle of viewpoints (ITU-T Recommendations X901 to X905 | ISO/IEC 10746), and use UML (Unified Modeling Language - http://www.rational.com/uml) object-oriented methodology notations.

Turning to Figure 1, which represents an enterprise viewpoint of the system, a plurality of Actors and Use Cases are shown. Before describing the Actors and Use Cases in the enterprise viewpoint, it will be helpful to define the meaning of certain terms used in this disclosure, as follows:

Glossary

The "context" of a bid is the set of values that the Bidders need to know in order to calculate their bids accordingly. For example, in the AARS application of this invention, the context can consist of: destination, time of day, day of week, type of call (voice/data), call forecast (to help calculate volume discounts). The values of these parameters are preferably discrete in order to limit the number of entries in the cache. Accordingly, the 'discretization' of some values may be required to be performed, either manually or according to well-known machine learning algorithms. For example, a continuum can be broken into a finite set of contiguous intervals of equal length. Thus, a variable of range [0..3] becomes a discrete variable with the following elements: '[0..1]', '[1..2]', '[2..3]'. A value such as 1.45 would be then rounded and assimilated as the element '[1..2]'.

A "bidding policy" defines the manner in which a Bidder calculates a bid given the context of the bid. A bidding policy can be represented in several ways. It can be as simple as a table containing all possible bid values, or it can be more complex, using a set of rules, or a spreadsheet. An example of a rule in the AARS context can be: "if time of day = PM then divide current bid value by 2". For the purpose of this disclosure, no assumptions are made regarding the way bidding policies are represented in the Bidders.

Actors

The Bid Manager: The Bid Manager is a software agent that calls for bids and selects the best bid from a plurality of Bidders. An example of a Bid Manager in U.S. Patent 5,675,636 is the assessment agent.

The Bidder: The Bidder is a software agent that, upon receipt of a call for bids, sends a bid for a given service that it would provide if it is selected. An example of a Bidder in U.S. Patent 5,675,636 is the costing agent.

Use cases

Cache bid: This use case starts when the Bid Manager receives a bid from a Bidder. The Bid Manager stores the information for a particular context of the call for bid, for later use.

Use cached bid: This use case starts when the Bid Manager wants to make a call for bids. If the context of that call is identical to one that has occurred previously, the Bid Manager will consult the cached information for the Bidders that have been contacted in the previous call instead of using the bidding protocol.

Update bid: This use case starts when a change occurs in a given Bidder's bidding policy. If the Bid Manager has subscribed to changes, the Bidder sends an update notification. The subscription to changes can be stored in a subscription list or table by the Bidder whenever it receives a call for bid. Each subscription corresponds to one context. When there is a change in the bidding policy, the Bidder calculates the new bid values for each of the subscribed contexts, and sends the new values to the Bid Manager. This can be a long process if the subscription list becomes too long. Accordingly, there are two solutions to solving this problem. In the first solution, the Bid Manager only subscribes for (and therefore only caches) the most used contexts in a subscription list (e.g. the first "n" entries), and unsubscribes to the others. For the others, the Bid Manager calls for bids. Another solution requires that the Bidder simply clears the subscription list and starts all over again with an empty list. This solution may also be used if the impact of the bid policy change is significant (e.g. if more than a certain percentage of the most used entries have been changed), wherein it would be inefficient to update all entries.

Turning to Figure 2, a class diagram is provided comprising a general representation of the information viewpoint of the system, with emphasis on the resource management and selection aspects.

The class diagram includes a class for ResourceManager, ResourceAdapter and Feature, as follows:

Classes

The Feature class represents a portion of the logic of an agent. It uses the ResourceManager to select ResourceAdapters to trigger actions on its environment.

The ResourceManager class manages a set of ResourceAdapters, and is responsible for selecting them.

The ResourceAdapter is a class that provides a uniform interface to access APIs of resources.

Turning to Figures 2A and 2B, the general class design of Figure 2 is applied to the particular case of the BidderAgentAdapter which is an interface for

communicating with the Bidder and use its services (Figure 2A) and which implements the getBidValue() method via one of either the CachingAdapter or the NoCachingAdapter (Figure 2B).

In these Figures, a notation has been added to the classical UML representation in order to represent Design Patterns (cd GoF book: Design Patterns: Elements of Reusable Object-Oriented Software, Gamma et al, Addison Wesley). Object-Oriented Design Patterns represent reusable pieces of object-oriented design that have been successfully used in different contexts in the past. The name in the dashed ellipse represents the name of the pattern, and the dashed lines represent the roles played by the different classes in that pattern. Thus, in Figure 2A, the BidderAgentAdapter plays the role of the *adapter*.

According to the Strategy Pattern set forth in Figure 2B, the CachingAdapter and NoCachingAdapter are used to implement the concrete getBidValue() method depending on whether caching is or is not to be used. The choice of whether the CachingAdapter or NoCachingAdapter is to be used may be made either by the BidManager or the Bidder. If the Bidder makes the decision on which Adapter is to be used, the Bidder can send the chosen Adapter to the BidManager for use (using for example the serialization and class loading mechanism present in the Java language. Similar mechanisms exist in other programming languages).The ResourceManager uses the getBidValue() method, irrespective of how the method is implemented in the subclasses.

Figure 3 is a diagram showing how, in the Distributed AARS example, a choice is made between use of the CachingAdapter or the NoCachingAdapter depending on the location of the Carrier Agents.

A shown in Figure 4, the Adapter that is not using caching (in this example, the NoCachingAdapter), sends a message to the Bidder to ask for its bid value. The Adapter then returns this value to the ResourceManager. The CachingAdapter first looks at its cache to see if it has already stored a bid value for the corresponding context in which case it returns that bid value. Otherwise, the CachingAdapter behaves like the NoCachingAdapter. Eventually, the CachingAdapter stores the new value in its cache for further reuse.

Figure 5 shows how the ResourceManager, which is the Client of the Strategy Pattern, selects an adapter by first sorting the Adapters by decreasing value of the

bids, and then selecting the first available Bidder. By doing so, two goals are achieved. First, there is no need to reserve the Bidders during the sorting phase. If a bid is not interesting to the BidManager, the Bidder is still free to answer other calls for bids sent by other BidManagers. The reservation of the Bidder does not occur until the Bid Manager is actually selecting the first available Bidder from the sorted list. Second, the sorted list of Bidders can be cached for a given context, thereby completely avoiding the bidding process in some cases.

Turning to Figures 6 to 10, interaction diagrams are provided which describe all possible application scenarios wherein one Bidder uses the caching mechanism of the present invention (Bidder B) and one Bidder does not use the caching mechanism (Bidder A).

The first diagram (Figure 6) shows the Bidders deciding which kind of ResourceAdapter they want the Bid Manager to use for the bidding protocol (i.e. whether or not to use caching). The request for service message asks agents if they want to take part in the subsequent bidding process. The Bidders agree to a request for service message by sending their selected Adapters.

Figure 7 shows the case where, for a given context, the Bid Manager does not have any previous bid from Bidder B in its cache. The cache is maintained and consulted by the Bid Manager. The Bid Manager executes the getBidValue(context 1) method via the NoCachingAdapter sent by Bidder A and the Caching Adapter sent by Bidder B. Thus, the NoCachingAdapter calls for a bid from Bidder A for the given context (context 1), in response to which Bidder A returns a bid (Bid 1) to the Bid Manager. Meanwhile, the Caching Adapter also calls for a bid from Bidder B which, in response, subscribes to the Update Bid use case for the given context (context 1) and returns a bid to the Bid manager (Bid 2). Accordingly, the Bid Manager now has cached the Bid 2 value for context1 from Bidder B so that the next time it implements the getBidValue(context 1) method, it will not send a call for bids to Bidder B, but will instead use its cached value.

This is shown in Figure 8 where, in response to a second request for bids for context 1, the NoCaching Adapter sends a call for bids to Bidder A which, as before, returns a bid which is forwarded to the Bid Manager. However, the CachingAdapter returns the previously cached bid (Bid 2) to the Bid Manager instead of call for a bid from Bidder B. As indicated above, the Bid Manager maintains the cache using The

CachingAdapter which is a component of the Bid Manager (having been sent by the Bidder to the Bid Manager).

In the event that the Bid Manager needs bids in a different context, it will be required to send a call for bid to Bidder B since there is no cached bid value for the new context. This is shown in Figure 9. The Bid Manager executes the getBidValue(context 2) method via the NoCachingAdapter sent by Bidder A and the Caching Adapter sent by Bidder B. Thus, the NoCachingAdapter calls for a bid from Bidder A for the given context (context 2), in response to which Bidder A returns a bid (Bid 3) to the Bid Manager. Meanwhile, the Caching Adapter also calls for a bid from Bidder B which, in response, subscribes to the Update Bid use case for the given context (context 2) and returns a bid to the Bid manager (Bid 4). Accordingly, the Bid Manager now has cached the Bid 4 value for the new context (context 2) from Bidder B so that the next time it implements the getBidValue(context 2) method, it will not send a call for bid to Bidder B, but instead use its cached value.

If Bidder B changes its bidding policies, it must notify the Bid Manager(s) of the change, in order to update the cache (Figure 10). If the bidding policies are, for example, in the simple form of entries in a table, Bidder B can just send the new value(s) of the entry or entries that have changed, so that the Bid Manager simply updates the cache. In cases where bidding policies are represented as general rules, a notification of change is made in the policies so that the Bid Manager deletes all entries corresponding to B, in order to restart caching. The following pseudo-code exemplifies the foregoing:

Case 1: policy represented as an entry in the table:

Bidder:

```
If entry-changed then
 Forall subscribers for the entry send new entry
value
```

Case 2: if policy represented in any other way (such as rules):

Bidder:

```
Forall subscribers to the bidder send change
notification
```

BidManager:

```
If received change notification then delete all
entries in the bidder's cache
```

Figure 11 is a flowchart showing the caching process according to the present invention. The process of Figure 11 will be better understood by considering the exemplary application of Figure 12 which shows a distributed Advanced Automatic Route Selection (AARS), wherein Carrier Agent B is remote from the Router Agent (i.e. the Bid Manager) and Carrier Agent A, and is connected to the Router Agent via a leased line. Thus, there would be a considerable cost in terms of time-efficiency if Carrier Agent B was required to send a bid to the Router Agent for each call. On the other hand, since Carrier Agent A is running in the same machine as the Router Agent, the caching mechanism does not need to be used for it.

The following scenario will illustrate the use of the invention:

- A call request is issued (new request in Figure 11) from a Line Agent (see U.S. Patent 5,675,636) with the following parameters representing the context of the call: (Destination: 613 TimeOfDay: AM). It will be appreciated that other parameters can be included such as DayOfWeek, TypeOfCall (voice or data), UsageForecast (to get volume discounts). However, for the sake of clarity this example considers only the use of two parameters. As discussed briefly above, in order to make the caching efficient, the values of the parameters should be discrete or discretized manually or by using a machine learning algorithm (e.g. TimeOfDay is represented in this example by two values: AM and PM). The content of the cache for Carrier B at this stage is: <empty>
- The router sends a call for bid (Figure 11) via the following message to Carriers A and B:
  (type: CallForBid context: (Destination: 613 TimeOfDay: AM))
- Carrier A replies with:
  (type: Bid value: 12)
- Carrier B replies with

(type: Bid value: 25)

The content of the cache becomes:

Context	BidValue
Dest: 613 ToD: AM	25

- The next call request (new request in Figure 11) has the following values for its parameters: (Destination: 613 TimeOfDay: PM)
- Since these values do not correspond to any entry in the cache (i.e. "No" following "context in cache?" in Figure 11), the router sends a call for bids to both Carriers A and B (call for bid in Figure 11).
- Carrier A replies with:
  (type: Bid value: 12)
- Carrier B replies with
  (type: Bid value: 19)

The contents of the cache is then updated (step 4) to become:

Context	BidValue
Dest: 613 ToD: AM	25
Dest: 613 ToD: PM	19

- The next call request (new request) has the following values for its parameters: (Destination: 613 TimeOfDay: PM)
- The router finds an entry corresponding to the context in the cache (i.e. "Yes" following "context in cache?" in Figure 11) , it therefore uses the cache value for Carrier B ("use cached value" in Figure 11), and only sends a call for bid to Carrier A. The cache is therefore not modified.
- In the event the Carrier B's bidding policy changes and the change affect only a few entries in the cache, Carrier B sends an update for the changes: (type: Update context: (Destination: 613 TimeOfDay: AM) value: 23).
  The content of the cache then becomes:

Context	BidValue

Dest: 613 ToD: AM	23
Dest: 613 ToD: PM	19

Otherwise, if the changes affect more than a predetermined user defined percentage of the entries (e.g. 20%), Carrier B can alternatively send a change notification to the Router Agent so that the Router simply deletes the cache altogether and starts caching again from scratch.

Alternatives and variations of the invention are possible. For example, although the example presented herein relates to bidding by costing agents to provide a long distance carrier server in an automatic route selection system, the principles of the invention have broad applications to many multi-agent systems where Bid Managers require the services of resources which are represented in the system by agents. For example, an airline reservation agent may tender bids from several airline ticketing agents whose bids may be cached for individual contexts (e.g. flight dates, fares classes, etc.). All such variations and applications are believed to be within the sphere and scope of the invention as defined by the claims appended hereto.